به نام خدا

دانشگاه تهران
دانشکده مهندسی برق و کامپیوتر

تمرین کامپیوتری ۱

شبکه‌های کامپیوتری

دکتر خونساری

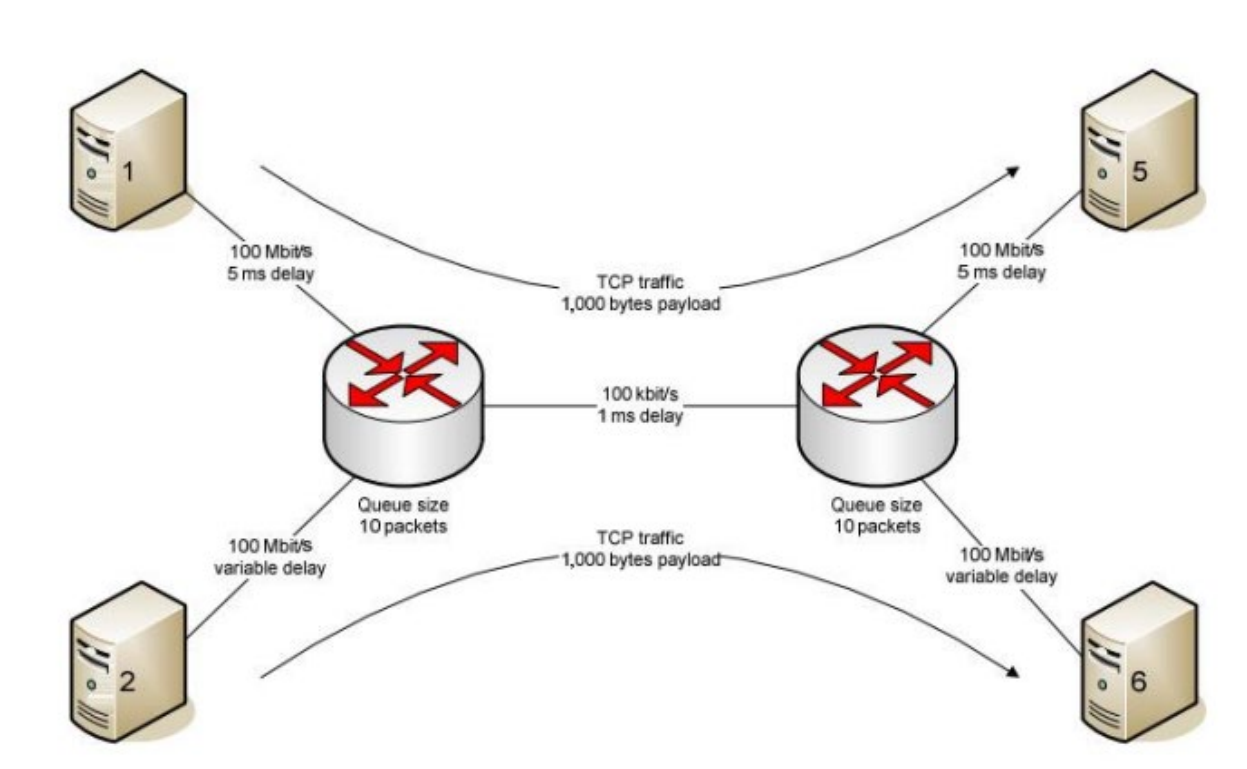
اعضای گروه:

- **علی قنبری** **۸۱۰۱۹۹۴۷۳**
- **اهورا شیری** **۸۱۰۱۹۸۵۵۵**

پاییز ۰۳-۱۴۰۲

۱.مقدمه

نرم افزار شبیه سازی NS2 یک ابزار قدرتمند در زمینه شبیه سازی شبکه های کامپیوتری و مخابراتی و همینطور رایانه ای با قابلیت پشتیبانی از پروتکل های مختلف شبکه است. در این پروژه به کمک نرم افزار نامبرده ، سعی داریم رفتار شبکه زیر را برای پیاده سازی های مختلف از پروتکل TCP بررسی نماییم.
در اینجا ۳ پیاده سازی New-Reno ، Vegas و Tahor مورد بررسی قرار میگیرد.



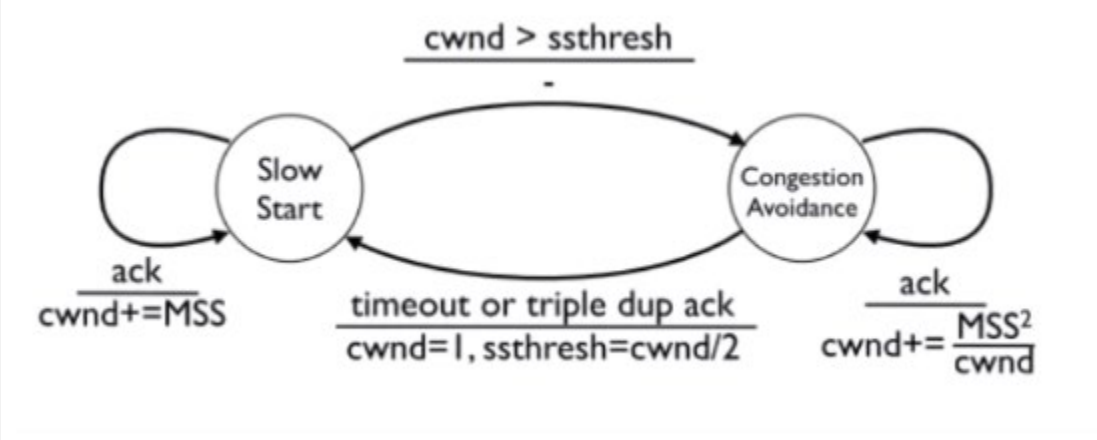
شکل ۱ : شبکه مورد بررسی در پروژه

برای تعریف توپولوژی شبکه و دریافت مقادیر ، از زبان TCL استفاده کردیم. NS2 علاوه بر TCL از C++ نیز پشتیبانی می کند اما در این پروژه از آن استفاده نمیکنیم.

۲.تعریف هر یک از پیاده سازی های TCP

TCP Tahoe ۱.۲

اینTCP، ۳ خاصیت را که عبارتند از start-slow, avoidance congestion و الگوریتم fast retransmit را به پروتکل TCP اضافه کرده است. اگر بازه ی مورد نظرِ Tahoe برای دریافت ack از سمت نود receiver پایان یابد یا ان که ۳ بار ack duplicate متوالی دریافت کند، آنگاه این چنین وضعیتی راpacket loss شناسایی میکند و بلافاصله بعد از تشخیص packet loss ، فاز fast retransmit خود را آغاز میکند که در این فاز فرستنده اول بسته loss شده را دوباره ارسال میکند و در گام بعدی مقدار ssthreshهمان(start-slow threshold)را به نصف اندازه ی فعلی پنجره مقدار دهی میکند و در نهایت فاز slow start- را با اندازه پنجره ۱ شروع میکند و در این فاز فرستنده با هر ack جدیدی که دریافت میکند اندازه پنجره اش را به صورت خطی و با سرعت نمایی افزایش میدهد .به طور کلی افزایش اندازه پنجره به صورت خطی و کاهش آن به صورت نمایی در مواجهه با packet loss یکی از ویژگی های tahoe است. یعنی tahoeبرای تشخیص ازدحام نیاز به مشاهده packet loss دارد.



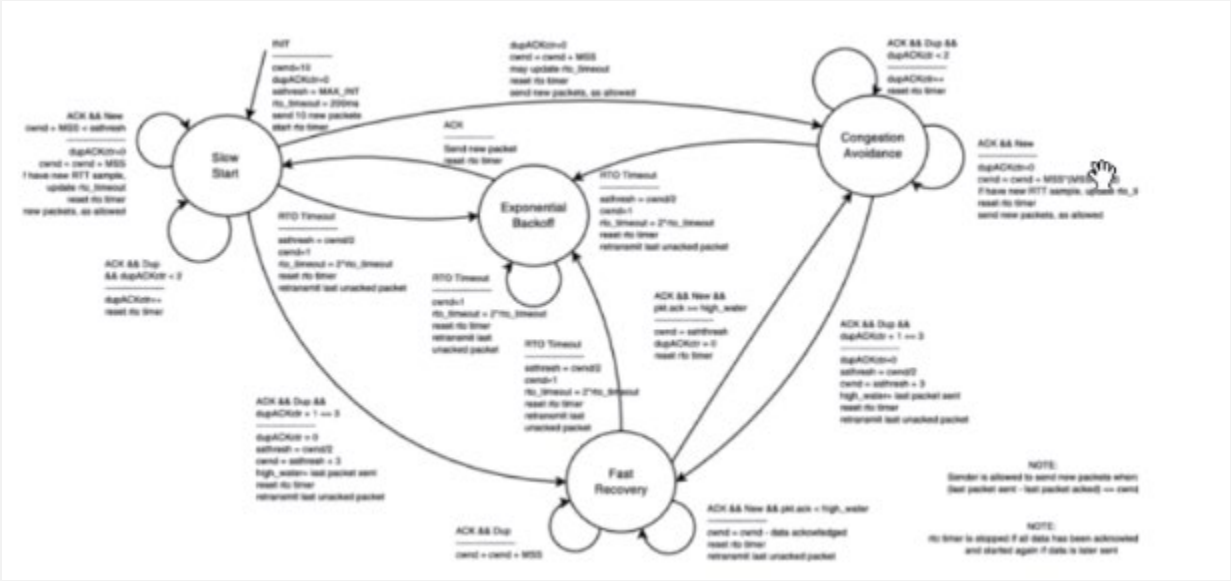
شکل ۲ : ماشین حالت برای Tahoe

TCP Newreno 2.2

این tcp فاز slow-start ، congestion avoidance و الگوریتم fast retransmit را مشابه tcp reno دارد.یعنی با شروع فاز start-slow در این tcp فرستنده با هر ack اندازه پنجره اش را یک واحد افزایش میدهد تا وقتی که به ssthresh برابر شود. وقتی برابر این مقدار شد، آنگاه فاز congestion avoidanceبه پایان میرسد. این tcp مشابه tahoe با دریافت ۳ بار duplicate ack متوالی ، این چنین وضعیتی را packet loss شناسایی میکند و بلافاصله بعد از تشخیص packet loss ، فاز fast restransmit خود را آغاز میکند و در این فاز اندازه پنجره اش و ssthresh هر دو به اندازه نصف اندازه فعلیِ پنجره مقدار دهی میشوند new reno .منتظر میماند تا ack تمامی بسته هایی که در مسیر هستند دریافت شوند. با این عملکرد، از retransmit duplicate جلوگیری میکند.

new reno مشابه tahoe برای تشخیص ازدحام، وابسته به دیدن packet loss است .

توضیح :duplicate retransmit نصف کردنِ مجدد اندازه پنجره در صورت مشاهده یک packet loss دیگر در فازfast retransmit می باشد.



شکل ۳ : ماشین حالت TCP Reno

TCP Vegas 3.2

Vegas از بقیه پیاده سازی های tcp بهتر عمل میکند. به این دلیل که این tcp سعی می کند ترافیک را قبل از گم شدن بسته تشخیص دهد. این tcp بیشتر از آنکه ترافیک را با packet loss تشخیص دهد، ترافیک را با مقدار rtt تشخیص میدهد. (که بر خلاف tcp های قبلی است که بررسی شدند).
این tcp ، ۳ ویژگی متفاوت نسبت به tcp های قبلی دارد

1 :این tcp زمان ارسال بسته را ثبت میکند و نیز یک تخمین از rtt نگه میدارد .وقتی یک duplicate packet می آید ، بررسی میکند که اگر از زمان ارسال آن تا زمان آمدن duplicate ack بیشتر از زمان تخمین زده شده برای rtt باشد؛ آن گاه بسته را دوباره میفرستد و در نتیجه منتظر دریافت 3 تا duplicate ack نخواهد ماند.

2 : برای مشخص کردن اندازه پنجره بر خلاف پیاده سازی های دیگر، بر اساس مقدارrtt کار میکند بر طبق فرمول های زیر :

if diff < α/base_rtt => cwnd(t + tA) = cwnd(t)+1

if α/base_rtt < diff < β/base_rtt => cwnd(t + tA) = cwnd(t)

if β/base_rtt < diff => cwnd(t + tA) = cwnd(t)-1

diff = cwnd(t)/ base_rtt – cwnd/rtt

base_rtt : کمترین مقدار rtt تا الان
rtt : مقدار واقعی rtt

۳: در فاز slow-start در هر rtt مقدار گذردهی لینک اندازه گرفته میشود و به صورت نمایی افزایش میابد تا جایی که تفاوت این مقدار با مقدار مورد انتظار از یک آستانه عبور میکند و در این زمان وارد فاز congestion avoidanceمیشود.

3. اسکریپت TCL استفاده شده در پروژه

در این قسمت کد های اسکریپت را توضیح میدهیم :

```
if { $argc != 1 } {
   puts "The congestion.tcl script requires tcp congestion algorithm name
as input"
   puts "For example, ns congestion.tcl Newreno"
   puts "Please try again."}
```

چک میکند آیا این اسکریپت با یک آرگومان اجرا شده است یا خیر. اگر با یک آرگومان چاپ نشده بود ، ارور بالا را چاپ میکند.

```
  if { [lindex $argv 0] == "Tahoe" } {
     set CONGESTION_ALGORITHM "TCP"
  } else {
     set CONGESTION_ALGORITHM "TCP/[lindex $argv 0]"}
```

اگر اسکریپت ، یک آرگومان دریافت کرد و چک میکند آیا پروتکل Tahoe هست یا خیر . و در صورت بودن یا نبودن ، مسیر مناسب را ست میکند.

```
set ns [new Simulator]
```

یک شی از کلاس Simulator ایجاد میکند. و به متغیر ns ، assign میکند.

```
set namfile [open congestion.nam w/
$ns namtrace-all $namfile
set tracefile [open congestion.tr w/
$ns trace-all $tracefile
```

دو فایل رهگیری (trace) را راه اندازی میکند، یکی برای nam congestion.nam و دیگری برای رهگیری عمومی شبیه سازی عمومی (congestion.tr). شبیه‌ساز (ns) برای ردیابی رویدادها در این فایل‌ها پیکربندی شده است و رکوردی از شبیه‌سازی برای تجزیه و تحلیل ارائه می‌کند.

```
proc finish{} {
    global ns namfile tracefile
$ns flush-trace
    close $namfile
    close $tracefile
    exit 0   }
```

تابع finish در پایان شبیه‌سازی شبکه فراخوانی می‌شود. اطلاعات ردیابی را پاک می کند، فایل های nam و ردیابی کلی را می بندد و سپس از اسکریپت خارج می شود.

```
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
```

۶ گره به ترتیب ۱ تا ۶ در شبکه تعریف میکنیم . (همان هاست ها هستند) گره ۳ و ۴ روتر هستند.

```
set rng [new RNG]
$rng seed 0
```

یک شی اعداد تصادفی (RNG) ایجاد می کند، seedآن را ۰ می کند و شی RNG را به متغیر rng اختصاص می دهد.

```
set n2_n3_delay [new RandomVariable/Uniform]
$n2_n3_delay set min_ 5
$n2_n3_delay set max_ 25
$n2_n3_delay use-rng $rng

set n4_n6_delay [new RandomVariable/Uniform]
$n4_n6_delay set min_ 5
$n4_n6_delay set max_ 25
$n4_n6_delay use-rng $rng
```

۲ متغیر تصادفی به نام n2_n3_delay و n4_n6_delay را با توزیع یکنواخت بین ۵ تا ۲۵ ایجاد میکند.

```
$ns duplex-link $n1 $n3 100Mb 5ms DropTail
$ns duplex-link $n2 $n3 100Mb [expr [$n2_n3_delay value]]ms DropTail
$ns duplex-link $n3 $n4 100kb 1ms DropTail
$ns duplex-link $n4 $n5 100Mb 5ms DropTail
$ns duplex-link $n4 $n6 100Mb [expr [$n4_n6_delay value]]ms DropTail
```

لینک های ۲ طرفه موجود بین گره هارا تعریف میکند (طبق مشخصات صورت پروژه ظرفیت ها و تاخیر آن ها انتخاب شده است) همچنین الگوریتم مدیریت صف ، **DropTail** می باشد.

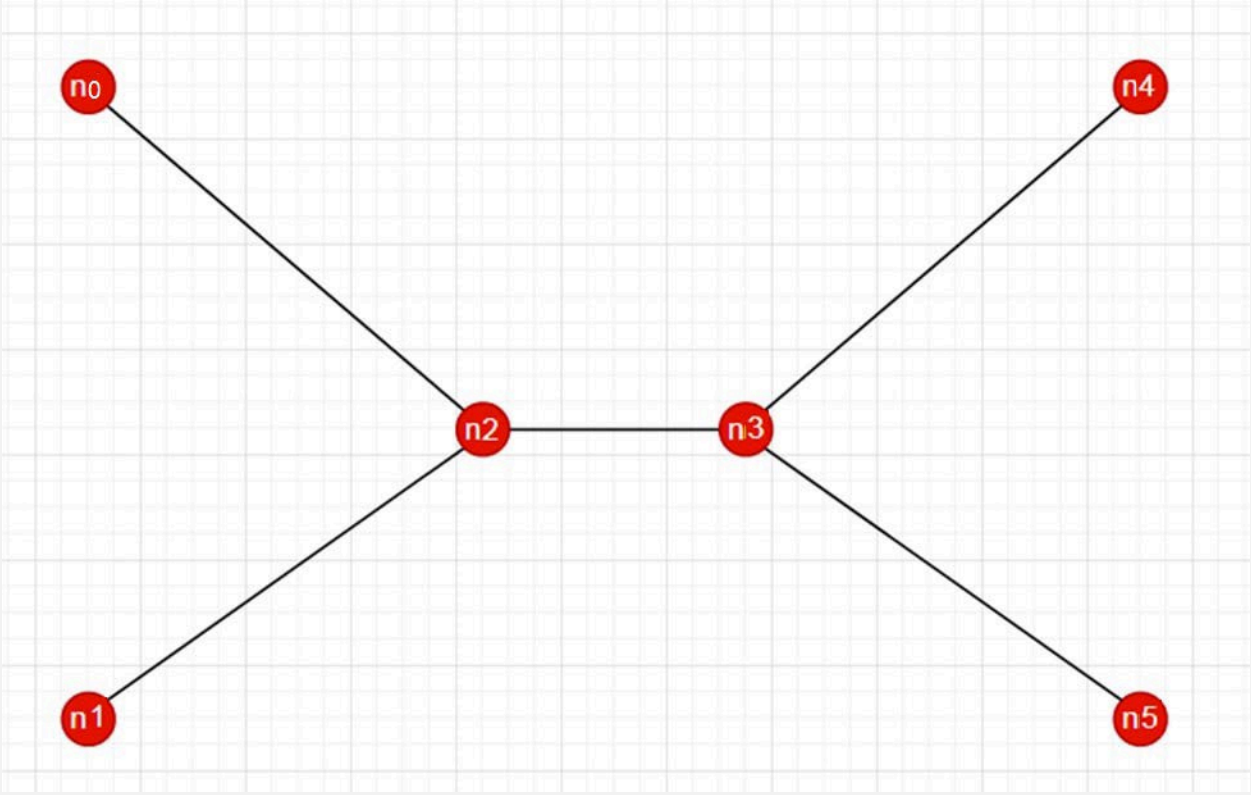
و برای لینک گره های ۲-۳ و ۴-۶ از متغیر تصادفی که در بالا تعریف شده بود ، استفاده کردیم.

```
$ns duplex-link-op $n1 $n3 orient right-down
$ns duplex-link-op $n2 $n3 orient right-up
$ns duplex-link-op $n3 $n4 orient right
$ns duplex-link-op $n4 $n5 orient right-up
$ns duplex-link-op $n4 $n6 orient right-down
$ns duplex-link-op $n3 $n4 queuePos 0.5
```

این دستورات برای تعیین پارامترهای عملیاتی اضافی برای لینک های دوطرفه در شبیه سازی شبکه، مانند جهت گیری و موقعیت صف. این پارامترها می توانند برای تجسم و تحلیل رفتار شبکه در طول شبیه سازی مهم باشند.

دستور آخر موقعیت صف لینک دو طرفه بین گره های n3 و n4 را روی ۰.۵ تنظیم می کند. این می تواند موقعیت صف را در امتداد لینک نشان دهد.

شکل (توپولوژی) شبکه ای که تعین کردیم به شکل زیر است:



```
$ns queue-limit $n3 $n1 10
$ns queue-limit $n3 $n2 10
$ns queue-limit $n3 $n4 10
$ns queue-limit $n4 $n3 10
$ns queue-limit $n4 $n5 10
$ns queue-limit $n4 $n5 10
$ns queue-limit $n4 $n6 10
```

سایز و محدودیت های صف ها لینک ها را تعریف میکنیم.

```
set tcp1 [new Agent/$CONGESTION_ALGORITHM]
$tcp1 set ttl_ 64
$tcp1 set fid_ 1
$ns attach-agent $n1 $tcp1
```

این کد یک عامل TCP با یک الگوریتم کنترل تراکم مشخص ایجاد می کند و آن را به گره **n1** در شبیه سازی شبکه متصل می کند.همچنین TTL بسته ها (Time-to-Live) به ۶۴ ست شده و نیز شناسه جریان آن (flow identifier = fid) به ۱ برای tcp1 ست شده است.

TTL برای محدود کردن تعداد hop هایی که یک بسته می تواند در یک شبکه طی کند ، استفاده می شود. خط آخر نیز عامل TCP به نام tcp1 را به گره n1 متصل میکند.عامل را به یک گره خاص در شبیه سازی ارتباط میدهد، که نشان می دهد که عامل در حال ارسال یا دریافت ترافیک از آن گره است.

```
set tcp2 [new Agent/$CONGESTION_ALGORITHM]
$tcp1 set ttl_ 64
$tcp1 set fid_ 2
$ns attach-agent $n2 $tcp2
```

این کد یک عامل TCP با یک الگوریتم کنترل تراکم مشخص ایجاد می کند و آن را به گره **n2** در شبیه سازی شبکه متصل می کند.همچنین TTL بسته ها (Time-to-Live) به ۶۴ ست شده و نیز شناسه جریان آن (flow identifier = fid) به 2 برای tcp2 ست شده است.

TTL برای محدود کردن تعداد hop هایی که یک بسته می تواند در یک شبکه طی کند ، استفاده می شود. خط آخر نیز عامل TCP به نام tcp2 را به گره n2 متصل میکند.عامل را به یک گره خاص در شبیه سازی ارتباط میدهد، که نشان می دهد که عامل در حال ارسال یا دریافت ترافیک از آن گره است.

```
set sink1 [new Agent/TCPSink]
$ns attach-agent $n5 $sink1
set sink2 [new Agent/TCPSink]
$ns attach-agent $n6 $sink2
```

سینک TCP مؤلفه ای است که برای دریافت و پردازش ترافیک TCP استفاده می شود، به عنوان مقصد یا نقطه پایانی برای ارتباط TCP عمل می کند. هدف اولیه TCP سینک ، دریافت بسته های TCP و ارائه مکانیزمی برای جمع آوری و تجزیه و تحلیل داده ها است.

در این کد گره های نهایی (مقصد) توسط عامل TCP تعین میشوند (یعنی گره n5 و گره n6)

```
$ns connect $tcp1 $sink1
$ns connect $tcp2 $sink2
```

در این کد بین فرستنده و گیرنده ترافیک با پروتکل tcp برقرار میشود. در حقیقت tcp agent ها یا عامل های tcp به سینک ها (گیرنده ها) وصل میشود.

```
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
```

یک کاربرد FTP به نام ftp1 ایجاد میکنیم . سپس این کاربرد را به عامل tcp (tcp1) مرتبط میکنیم . در حقیقت داریم FTPرا روی TCP برای ارتباطات آن ، سوار میکنیم. برای ftp2 نیز همین سناریو را انجام میدهیم.

```
$ns at 0.0 "$ftp1 start"
$ns at 0.0 "$ftp2 start"
$ns at 1000.0 "$ftp1 stop"
$ns at 1000.0 "$ftp2 stop"
$ns at 1000.0 "finish"
```

کد بالا برنامه ریزی جریان شبکه را انجام میدهد.متود start کاربرد ftp1 درزمان ۰.۰ اجرا میشود. یعنی ftp1 در زمان ۰ شروع به کار میکند.برای ftp2 نیز همینطور است. در زمان ۱۰۰۰ نیز ۲ کاربرد بالا متوقف میشوند و ترافیک ها از بین میروند.
در خط آخر ، تابع finish در زمان ۱۰۰۰ فراخوانده میشود. این تابع ، شبیه سازی را به پایان میرساند.

```
proc plotCwnd {tcpSource outfile} {
  global ns
  set now [$ns now]
  set cwnd_ [$tcpSource set cwnd_]

  puts  $outfile  "$now,$cwnd_"
  $ns at [expr $now + 1] "plotCwnd $tcpSource $outfile"
}

set cwndTcp1File [open "cwnd1.csv" w]
set cwndTcp2File [open "cwnd2.csv" w]
puts  $cwndTcp1File  "time,cwnd"
puts  $cwndTcp2File  "time,cwnd"
$ns at 0.0  "plotCwnd $tcp1 $cwndTcp1File"
$ns at 0.0  "plotCwnd $tcp2 $cwndTcp2File"
```

رسم داده های plotCwnd :

تابع (plotCwnd) را تعریف می کند و مقادیر پنجره ازدحام TCP را برای دو منبع tcp1 و tcp2 در شبیه سازی شبکه تنظیم می کند.

```
proc plotGoodput {tcpSource prevAck outfile} {
  global ns
  set now [$ns now]
  set ack [$tcpSource set ack_]

  puts  $outfile  "$now,[expr ($ack - $prevAck) * 8]"
  $ns at [expr $now + 1] "plotGoodput $tcpSource $ack $outfile"
}

set goodputTcp1File [open "goodput1.csv" w]
set goodputTcp2File [open "goodput2.csv" w]
puts  $goodputTcp1File  "time,goodput"
puts  $goodputTcp2File  "time,goodput"
$ns at 0.0  "plotGoodput $tcp1 0 $goodputTcp1File"
$ns at 0.0  "plotGoodput $tcp2 0 $goodputTcp2File"
```

رسم داده های goodput :

تابع plotGoodput برای گرفتن زمان شبیه‌سازی فعلی، مقدار تأیید (ack) یک منبع TCP مشخص (tcpSource) و مقدار تأیید قبلی (prevAck) تعریف شده است.

goodput (نرخ انتقال داده) را به عنوان تفاوت بین مقدار ACK فعلی و مقدار ACK قبلی ضرب در ۸ (برای تبدیل از بایت به بیت) محاسبه می کند.

این کد مکانیزمی را برای رسم و ثبت مداوم مقادیر goodput (نرخ انتقال داده) دو منبع tcp1 و tcp2 در طول شبیه سازی تنظیم می کند. تابع plotGoodput برنامه ریزی شده است که به صورت دوره ای فراخوانی شود (در ثانیه $now +1) و اطلاعات ضبط شده برای جداسازی فایل های خروجی (goodput1.csv و goodput2.csv) نوشته می شود.

```
proc plotRTT {tcpSource outfile} {
  global ns
  set now [$ns now]
  set rtt_ [$tcpSource set rtt_]

  puts  $outfile  "$now,$rtt_"
  $ns at [expr $now + 1] "plotRTT $tcpSource $outfile"
}

set rttTcp1File [open "rtt1.csv" w]
set rttTcp2File [open "rtt2.csv" w]
puts  $rttTcp1File  "time,rtt"
puts  $rttTcp2File  "time,rtt"
$ns at 0.0  "plotRTT $tcp1 $rttTcp1File"
$ns at 0.0  "plotRTT $tcp2 $rttTcp2File"
```

رسم داده های RTT :

این کد مکانیزمی را برای رسم و ثبت مقادیر پیوسته زمان رفت و برگشت (RTT) دو منبع tcp1 و tcp2 در طول شبیه سازی تنظیم می کند. رویه plotRTT برنامه ریزی شده است که به صورت دوره ای فراخوانی شود و اطلاعات ضبط شده برای جداسازی فایل های خروجی (rtt1.csv و rtt2.csv) نوشته می شود.

```
$ns run
```

اجرای شبیه سازی شبکه توسط ns2 . شبیه سازی تا زمانی که کامل شود یا تا زمانی که به زمان شبیه سازی مشخصی برسد اجرا می شود.

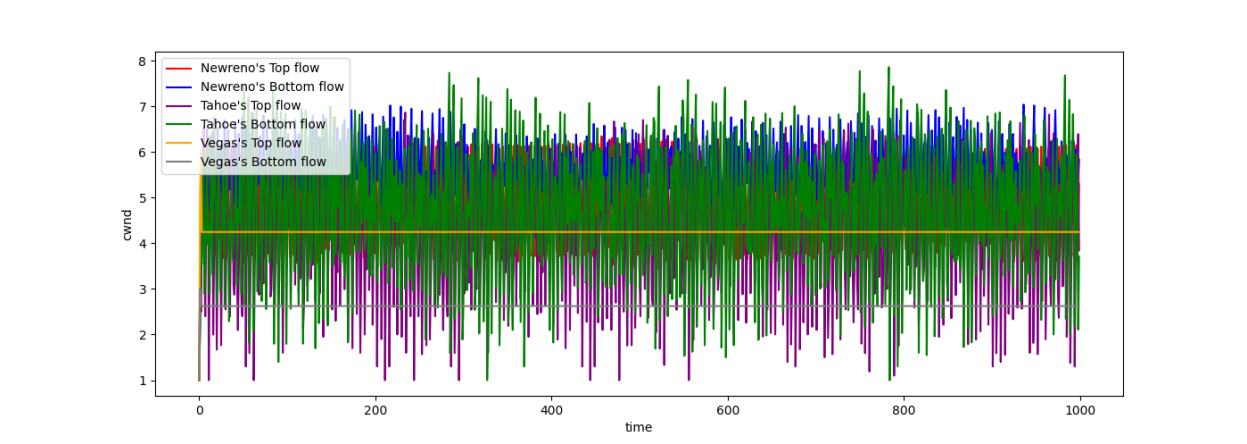
۴. نوتبوک پایتون تحلیل کننده نتایج

یک نوت بوک به نام congestion_analysis نوشتیم که وظیفه اجرای ۱۰ بار شبیه سازی شبکه و جمع آوری نتایج را بر عهده دارد. ۱۰ بار شبیه سازی انجام میشود سپس از نتایج میانگین گیری میشود و در نهایت نتایج در پوشه network simulation output قابل مشاهده است.

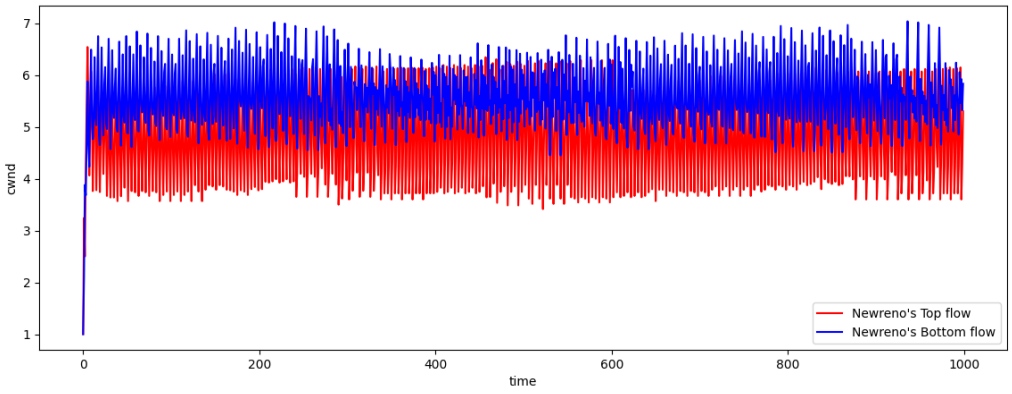
۵. نتایج اجرا

۵.۱ CWND

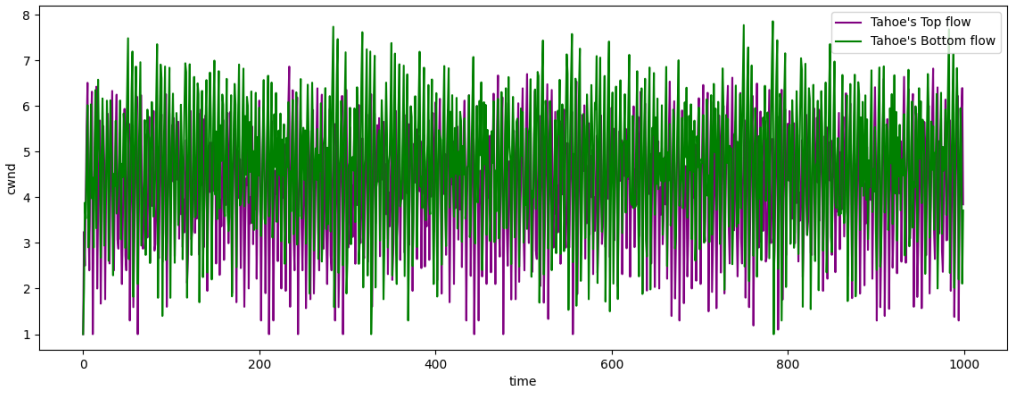
در تصویر زیر نتیجه حاصل از شبیه سازی و مقدار CWND برای تمامی اتصالات آمده است.در تمامی نمودار های زیر packet است.



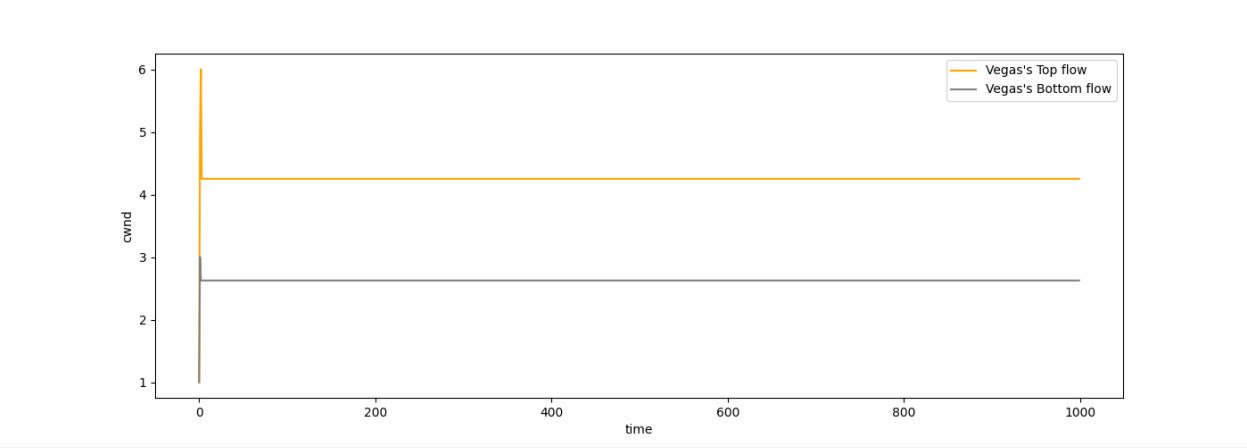
شکل ۴ : CWND همه اتصالات



شکل ۵ : CWND برای Newreno



شکل ۶ : CWND برای Tahoe

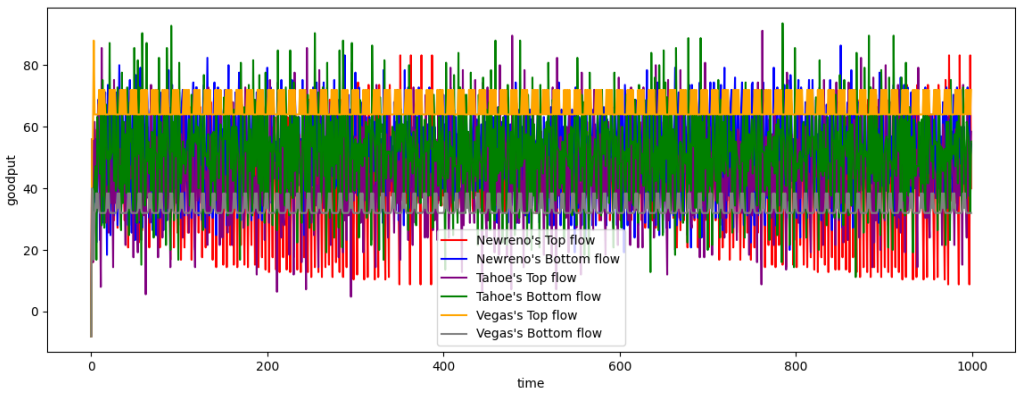


شکل ۷ : CWND برای Vegas

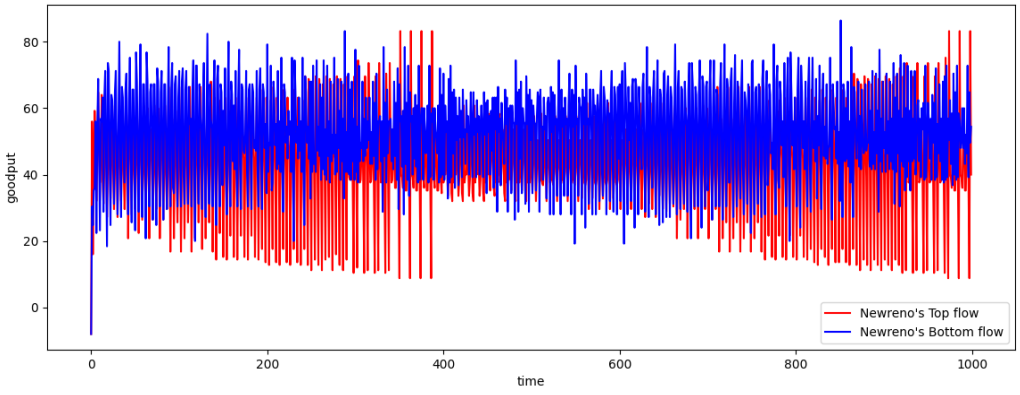
با مشاهده نمودار ها در میابیم که Newreno در مقایسه با Tahoe مقدار CWND مشخص تر و با نوسان کمتر دارد.به طور میانگین نیز مقدار آن بیشتر است. برای Vegas نیز پس از مدت کوتاهی مقدار CWND به عدد مشخصی همگرا شده که ناشی از فرآیند های پیشبینی است که در الگوریتم رخ می دهد. این فرآیند سبب می شود که الگوریتم استفاده شده از طریق پیشبینی RTT نرخ ارسال خود که به CWND وابسته است را به گونه ای تنظیم میکند که پس از مدتی به عدد مشخص همگرا می شود و تضمین می شود که افت پکت رخ نمیدهد.

goodput ۵.۲

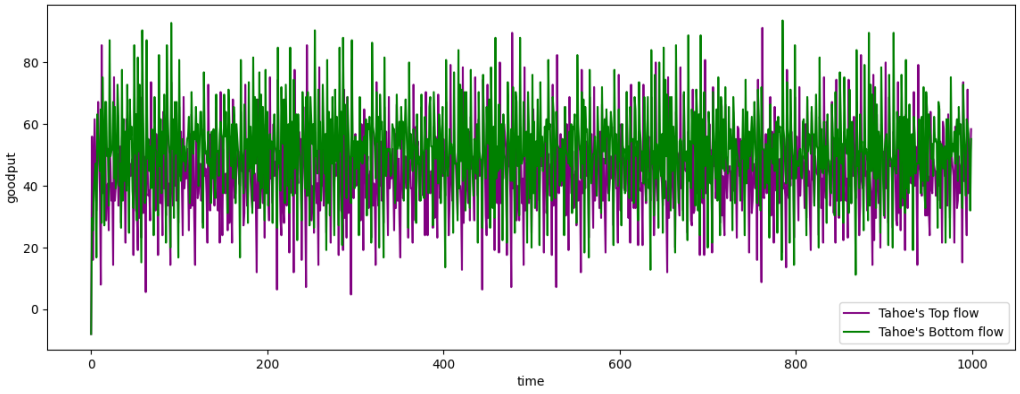
در تصویر زیر نتیجه حاصل از شبیه سازی و مقدار goodput برای تمامی اتصالات آمده است.واحد اندازه گیری مگابیت بر ثانیه است.



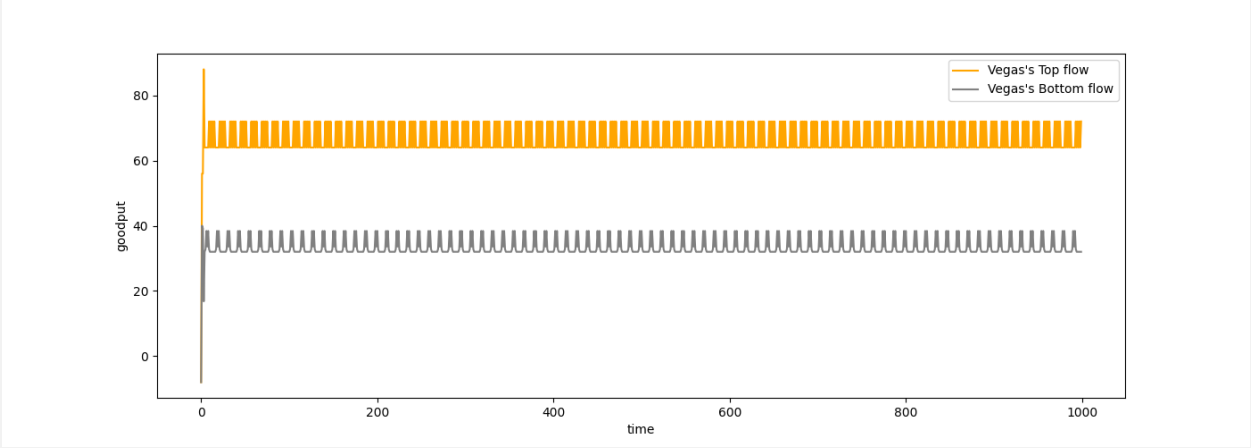
شکل ۸ : goodput همه اتصالات



شکل ۹ : goodput برای Newreno



شکل ۱۰ : goodput برای Tahoe

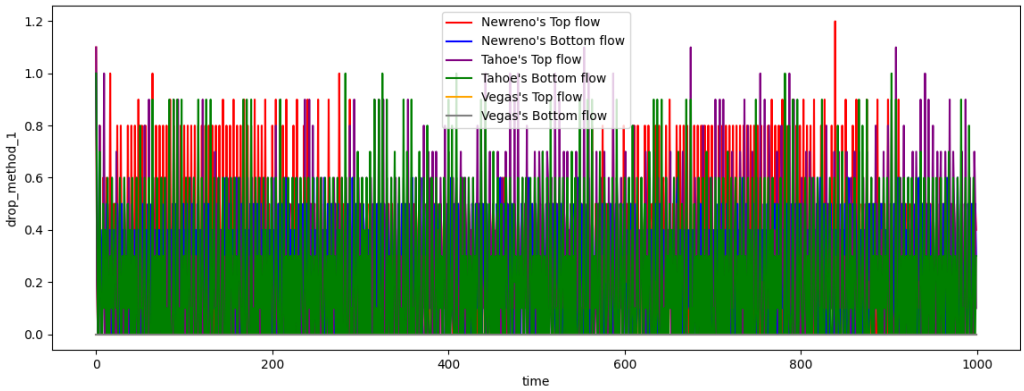


شکل ۱۱ : goodput برای Vegas

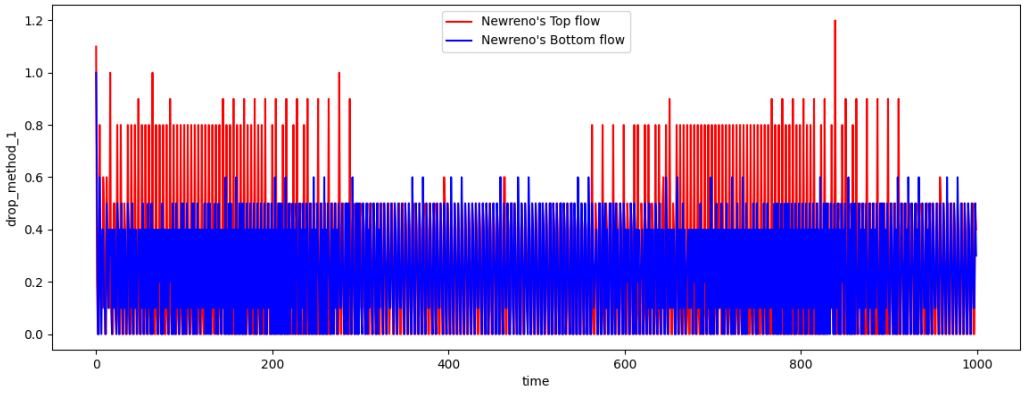
بازه تغیرات goodput برای Vegas نسبت به سایر ، کوچک تر است که علت آن ثابت بودن نرخ ارسال داده در Vegas است . در مورد Tahoe نرخ نوسانات بیشتر از بقیه است که نشان دهنده این است که شبکه بیشتر با Congestion درگیر است و درنتیجه goodput روند ثابتی را طی نمیکند.

Drop rate ۵.۳

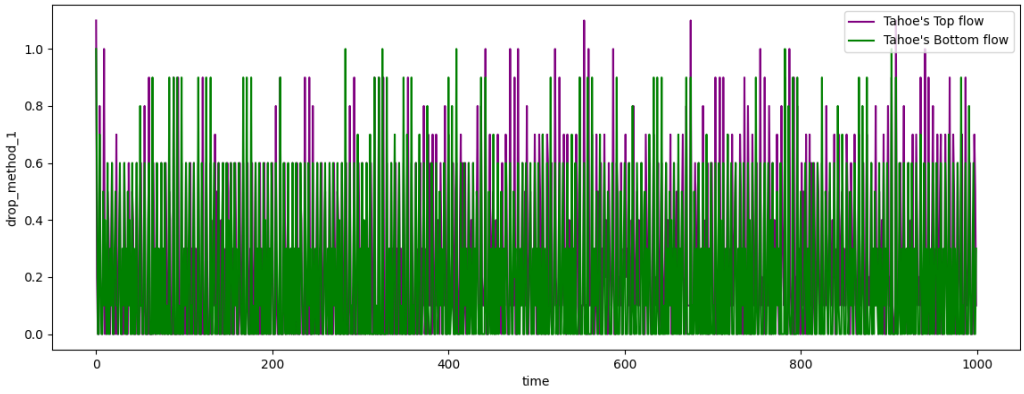
با توجه به این که معیار Drop rate یک معیار تجمعی است ، در این جا ما باید این معیار را برای هر ثانیه نشان دهیم که این امر سبب ایجاد پیچیدگی هایی مانند این که بسته drop شده در یک زمان مربوط به محاسبه برای زمان drop شدن است یا زمان ارسال و از این دست پیچیدگی هایی که سبب می شد تعریف این معیار ممکن نباشد ، فرض کردیم درخواست شما تعداد بسته های drop شده در واحد زمان برای هر ارتباط است.



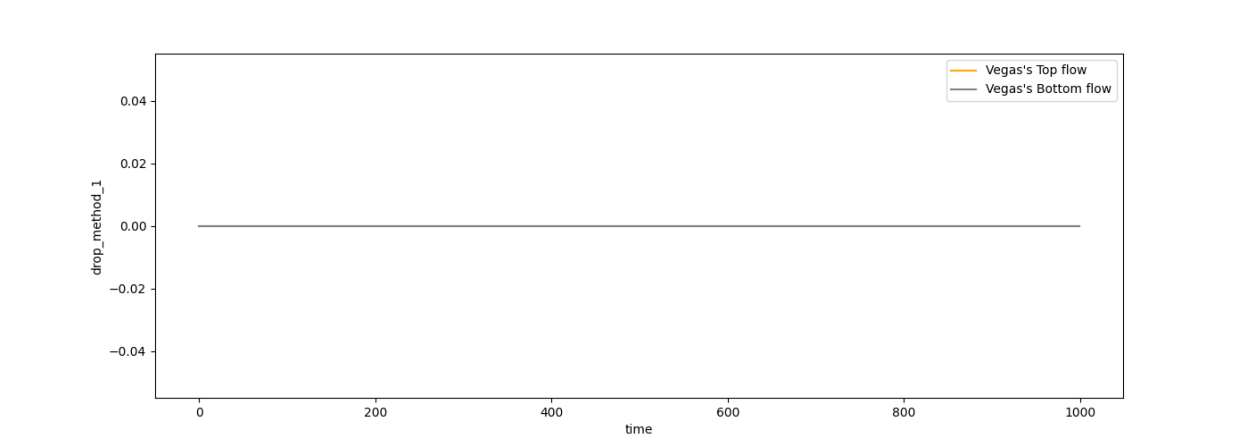
شکل ۱۲ : Drop rate همه اتصالات



شکل ۱۳ : Drop rate برای Newreno



شکل ۱۴ : Drop rate برای Tahoe

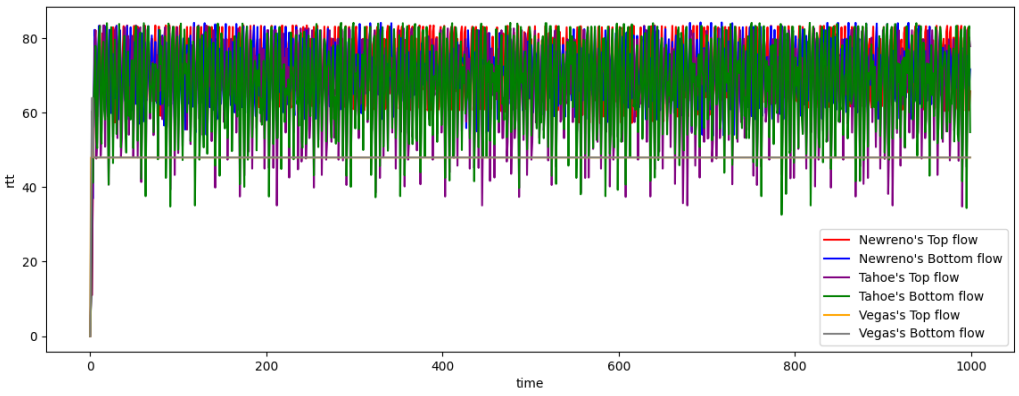


شکل ۱۵ : Drop rate برای Vegas (۲ نمودار روی هم)

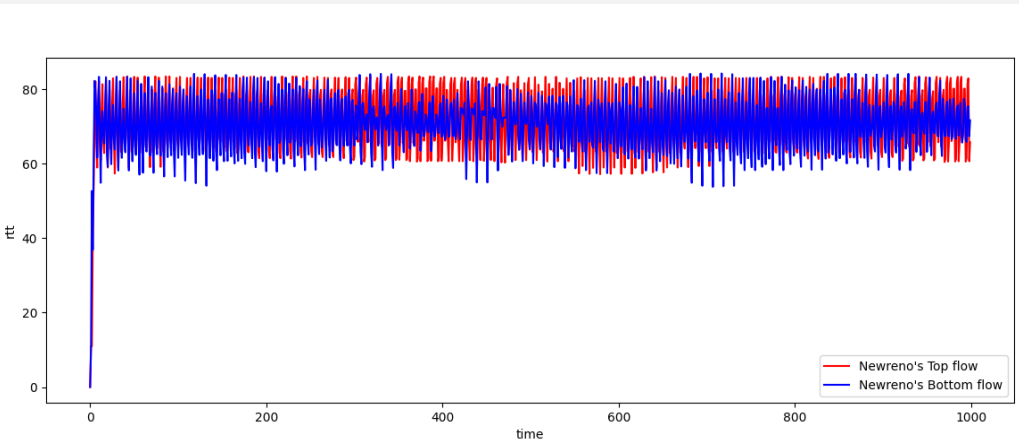
با توجه به شکل Vegas میتوان دریافت که فرآیند تشخیص زودهنگام سبب شده که این الگوریتم هیچ بسته ای را نداشته باشد که از دست رفته باشد. برتری این پروتکل قابل ملاحضه است.برای ۲ پیاده سازی دیگر نتایج به طور تقریبی یکسان است اما Newreno در برخی موارد بهتر عمل کرده است.

RTT ۵.۴

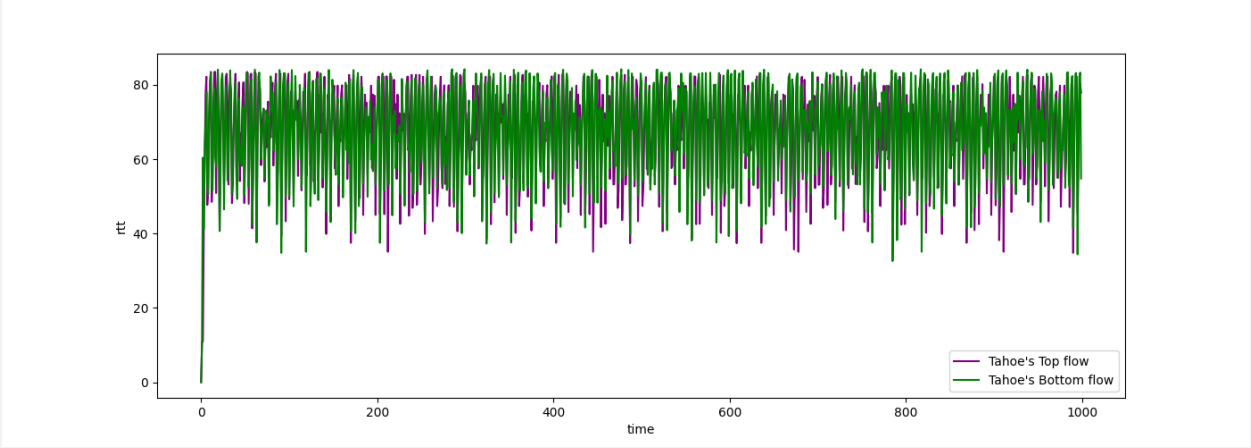
در تصاویر زیر نتیجه حاصل از شبیه سازی و مقدار RTT برای تمامی اتصالات آمده است.واحد RTT در نمودار های زیر ms است.



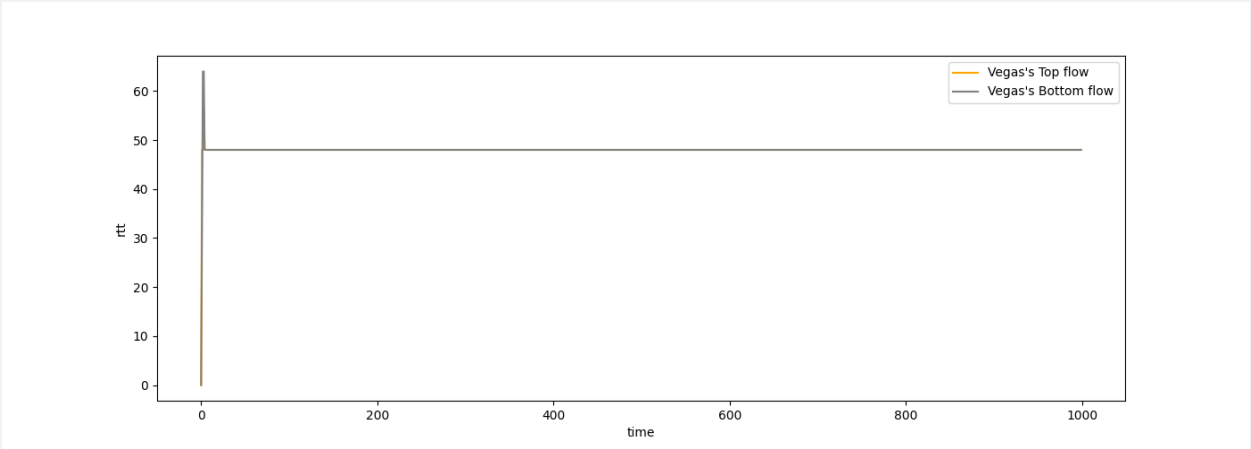
شکل ۱۶ : RTT همه اتصالات



شکل ۱۷ : RTT برای Newreno



شکل ۱۸ : RTT برای Tahoe



شکل ۱۹ : RTT برای Vegas (۲ نمودار روی هم)

چون Vegas به پیشبینی packet loss پیش از وقوع می پردازد ، بعد از مدت کوتاهی مقدار RTT برای آن ثابت شده است.با توجه به هدف اصلی Newreno که بهبود عملکرد Tahoe و Reno است ، دامنه تغیرات RTT در Newreno کمتر از Tahoe هست.که علت آن این است که در reno کنترل ازدحام سریع تر به فاز ازدحام وارد شود که باعث می گردد میزان ازدحام کمتر شده و در نتیجه بسته ها به طور متوسط RTT کمتری داشته باشند.